システムの記事

UMLを用いたモデル駆動開発の実践例

ETロボコン2006 Teamふるかわ参戦記

Teamふるかわ(佐藤 征宏，清水 正久，都 軍安，早坂 哲)

ここでは，UML(unified modeling language)モデリング・ツールやシミュレーション・ツールを利用したモデル駆動開発(MDD：model driven development)への取り組みについて解説する．UMLモデルを用いてコードを自動生成することにより，モデルとコードを完全に一致させた．また，モデルを実行環境から独立させて，同一のモデルをシミュレーションや論理テストにも利用できる環境を整えた．　(編集部)

筆者ら(Teamふるかわ)は，車載電子部品の組み込みソフトウェアを開発しています．車載電子部品の中でも主にボディ系と呼ばれる領域(エアコンやインテリジェント・キーなど，車を快適あるいは便利にする領域)を扱っています．この分野の組み込みソフトウェアも，ほかの分野の組み込みソフトウェアと同様に年々機能が増え，規模が大きくなっています．例えば，10年前には8ビットの1チップ・マイコンでROMサイズが32Kバイト，RAMサイズが1Kバイトで済んでいたエアコンが，今では16ビットの1チップ・マイコンでROMサイズが128Kバイト，RAMサイズが6Kバイトになっています．

これまではハードウェアだけで実現できていた部品にも，高機能化やネットワーク化に対応するためにマイコンと組み込みソフトウェアが搭載されるようになってきました．そのため，ソフトウェアの開発効率を改善することが急務になっています．

また，車に搭載するという性質上，品質は厳しく要求されます．例えば，コード・サイズ(マイコンの場合ROMサイズと等価)が4倍になったら，単位コードあたりの品質を4倍に高めないと，ハードウェアと組み合わせた電子部品としての品質は低下することになってしまいます．このようなことは当然ながら許されません．従って，開発効率と品質の両方を大幅に改善しなければならない，というのが現在の筆者らの状況です．そして，そのための切り札の一つが，モデル駆動開発(MDD：model driven development)ではないかと考えています(p.142のコラム「これまでのモデル駆動開発への取り組み」を参照)．

筆者らはすでに，スウェーデンTelelogic社のStatemate(構造化)，米国The Math Works社のMATLAB/Simulink(システム・シミュレーション)といった，モデル駆動開発に対応したツールを使っています．これにUML(オブジェクト指向)を加えて，モデリング技術を適材適所に使い分けるようになりたいという目標を持ちました．そのために，UMLを習得するきっかけとして，2006年，UMLなどを用いたロボット・コンテスト「ETソフトウェアデザインロボットコンテスト(ETロボコン)」に参加しました．

● 自動コード生成でモデルとコードを完全一致させる

基本方針として，「モデルから自動でコードを生成する」を掲げました(**図1**)．モデルから自動でコードを生成できるメリットは，単にコーディング作業が減るという以外に，モデルとコードが完全一致することが非常に大きいと思います．これにより，モデルとコードが一致しているかどうかを確認するためのテストや検証が不要になります．また，チーム内でのレビューやコミュニケーションがモデルだけ

KeyWord　モデル駆動開発，UML，ETロボコン，MATLAB/Simulink，Rhapsody，シミュレーション

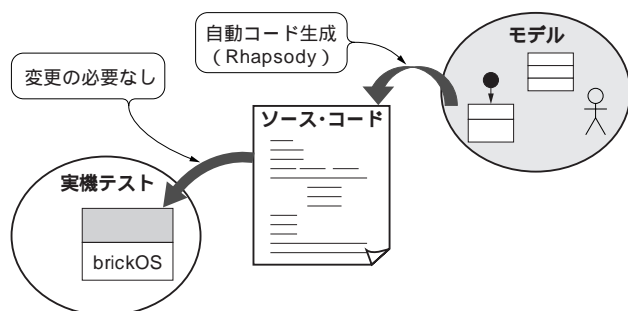



図1　基本方針は「自動コード生成」
モデルとソース・コードが完全に一致していれば，コーディング作業だけでなく，テストやレビューなどにも役立つ．

でできるようになります．良いモデルさえあれば，森を見たいときには森（UMLで言うとクラス図など）を，葉を見たいときには葉（UMLで言うとステートチャートなど）を瞬時に見られるようになります．

さらに，どんなに仕様変更があったとしても，モデルのほうを変更する（コーディングはツールに任せる）ことを徹底するだけで，モデルとコードの一致が維持されます．誰がメンテナンスしようとも，モデルとコードが一致したままの状態が維持されるというのは，モデル駆動開発以前の開発と比べると夢のような世界です[注1]．

注1：組み込みソフトウェアの場合，仕様変更やハードウェアの制約をソフトウェアでカバーするのが世の常となっている．これらのイベントが開発終盤に発生しようものなら，もはやUMLの図を描き直したりすることはなく，直接コードに手を出してしまうというのは，よく耳にする話である．自動コード生成が当たり前になれば，この状況も変わるだろう…．

● RhapsodyでC言語のソース・コードを生成

今回，ソフトウェア・モデリング用のUMLモデリング・ツールとして，Telelogic社のRhapsodyを使用しました．また，ハードウェアまで含めたシステム全体のシミュレーション用として，MATLAB/Simulinkを使用しました．

RhapsodyはUMLモデルからC言語，C++言語，Java言語のソース・コードを生成できます（ETロボコンの開発環境も，この三つの言語に対応している）．C言語を選択するとツールとして制約が発生し，オブジェクト指向らしいモデリングが難しくなります．しかし，筆者らが実務で使用している8ビットや16ビットのマイコンではC言語しか使えず，実務で使えない言語を選択したのでは勉強にならないため，あえてC言語を選択しました．

● アダプタを用意してOSに依存しないモデルを構築

ソフトウェアの基本構造としては，モデルから自動生成したソース・コードが，ハードウェアやリアルタイムOSなどの実行環境に依存しないようにしたいと考えました．

コラム　これまでのモデル駆動開発への取り組み

モデル駆動開発が銀の弾丸である（それですべてうまく行く）とは考えていませんが，モデル駆動開発にはかなり前から期待していました．「開発言語がアセンブリ言語からC言語に変わったときのような大きな効果が，C言語からモデル言語に変えることで得られるのではないか」と考えたのです．筆者らは，10年くらい前までは組み込みソフトウェアをアセンブリ言語で開発していました．ある時からC言語が使えるようになって，開発効率と品質の両方が大きく向上したという経験を持っています．モデル駆動開発への期待は，これと同じような体験がもう一度できるのではないかという期待から始まりました．そして，1990年代の終わりごろから，モデル・ベースのツールを少しずつ使い始めました．

筆者らが使っている主なモデル・ベースのツールは，ソフトウェア開発用としてはTelelogic社のStatemate，ハードウェアまで含めたシステムのシミュレーション用としてはThe MathWorks社のSimulinkです．

Statemateは，自動車業界でよく使われている（比較的）有名なツールであり，構造化分析/設計手法に基づいています．もともとは，仕様をモデル化するとシミュレーションが実現できることによる「動く仕様書を作成するための分析ツール」でしたが，設計ツールと一体化されて，組み込みソフトウェアのコードを自動生成できるようになっています．

Simulinkは，制御や信号処理の分野などで広く使われているツールです．微分方程式や差分方程式で表されるような数学モデルを制御ブロック図で記述できます．また，オプションでステートチャート（状態図）を記述したり，モデルから組み込み用のコードを自動生成したりする機能もあります．ドメインによってはSimulinkだけで開発するほうがよい場合もあると思いますが，筆者らはこれら二つのツールを使い分けています．

これらのツールにより，最近は量産開発にモデル駆動開発を適用できるようになりました．当初は，モデル駆動開発にそれほど苦労するとは考えていませんでしたが，ここにたどりつくまでに数年かかりました．アセンブリ言語からC言語への移行が比較的スムーズだったことを考えると，これは予想外の苦戦でした．しかし苦労はしたものの，やっとある程度使えるようになり，最近では効果も確実に出始めていると実感しています．

そうすれば，自動生成したコードを1行も変更せずに，米国Microsoft社のVisual C++のようなパソコン上の環境で論理テストを実施したり，Simulinkモデルに組み込んでシミュレーションしたりできます．

実際にロボットを動作させるには，モータやセンサといったハードウェアを制御する必要があります．今回使用するロボットの場合，デンマークLEGO社のLEGO MINDSTORMS用のオープン・ソースOS「brickOS」と，そのOSが持つAPI（application programming interface）を介して各種ハードウェアを制御することになります．このとき，モデルからbrickOSのAPIを直接呼び出すようにすると，そのモデルはbrickOSに依存したモデルになってしまいます［**図2（a）**］．

そこで，両者の情報を受け渡すためのアダプタを用意しました［**図2（b）**］．モデルは，直接実行環境に指示を出す（brickOSのAPIを呼び出す）のではなく，アダプタによって定期的に自分の状態を読み取ってもらい，アダプタが実行環境に指示を出します．また，実行環境側の刻一刻と変化する事象については，アダプタ経由で定期的に情報を与えてもらいます．そのために，モデルはアダプタに対してset/get関数を用意します．

このようにアダプタを用意することで，モデルが実行環境に依存することを完全に断ち切ることができます．これでVisual C++上でもSimulink上でも，モデルには一切手を加えずに，それぞれの環境に応じたアダプタを用意するだけで，あらゆる環境下で動作させることができます（**図3**）．

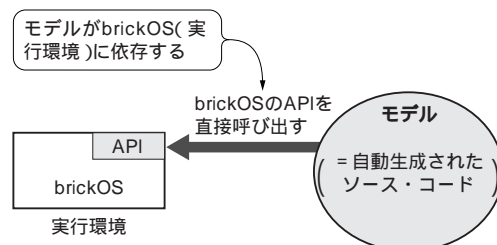
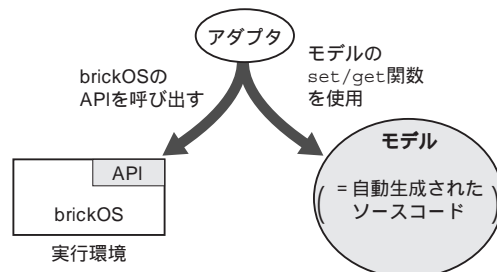
続いて，モデル側で用意するset/get関数について詳

（a）モデルは実行環境に依存する

（b）モデルは実行環境に依存しない

図2　アダプタを用意してモデルを実行環境から独立させた
実行環境に依存したモデルを別の環境下で動作させるためには，ポーティング（移植）作業が必要になってしまう．そこで，モデルと実行環境の情報を受け渡すためのアダプタを用意し，モデルを実行環境から独立させた．

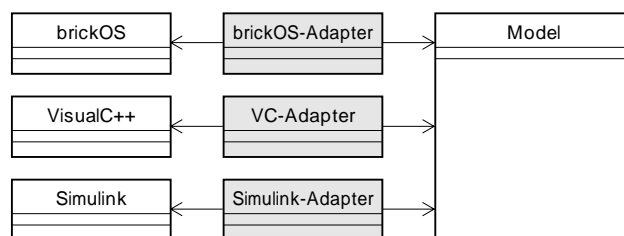



図3　アダプタを用意してモデルを複数の環境に対応させた（クラス図）
brickOS，Visual C++，Simulinkのそれぞれに対してアダプタを用意する．

PathFinder System
情報を受け取る系
タッチ・センサの値を受け取る
光センサの値を受け取る
情報を教える系
駆動モータの駆動方向を教える
駆動モータの速度を教える
首振りモータの駆動方向を教える
首振りモータの速度を教える
外部システム
VisualC++　brickOS　Simulink
どんな種類のアクタも同様のサービスが受けられる

図4
ライン・トレース・ロボットのユースケース図
ライン・トレース・ロボットとして，どのような外部とのインターフェースが考えられるかなどを整理した．ここで決定したインターフェースは，そのまま，後述するSimulinkとのインターフェース仕様書にもなる．

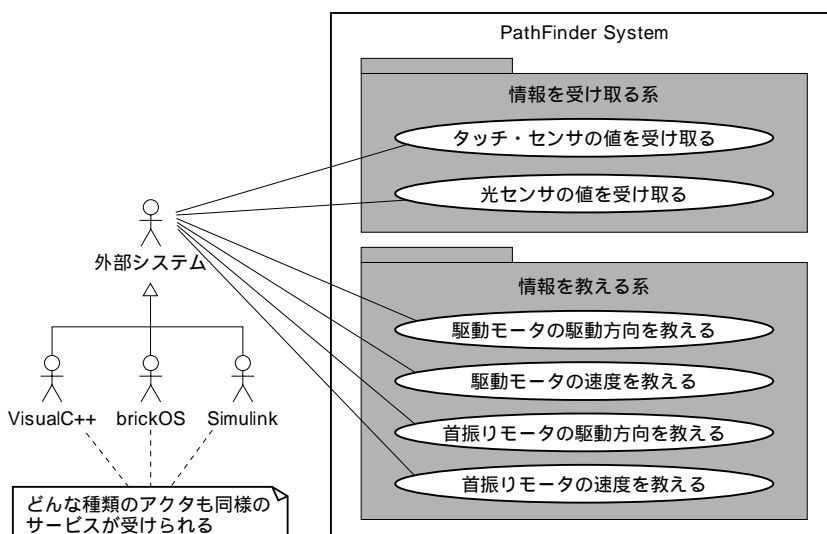
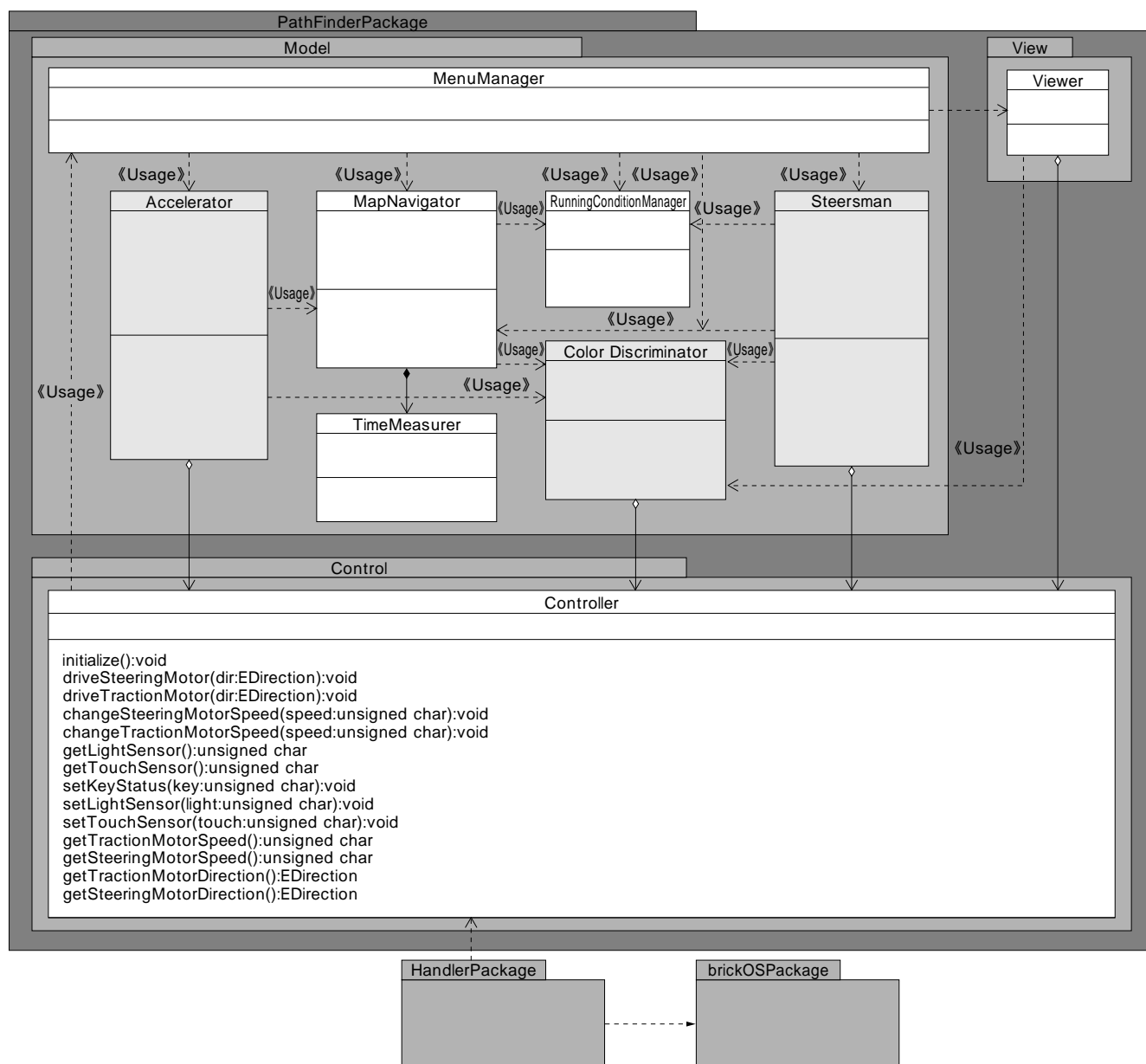



図5　ライン・トレース・ロボットのクラス図

HandlerPackageというのが，brickOSとモデル（PathFinderPackage）をつなぐアダプタである．なお，クラス内の変数や関数の記事は省略している（唯一，Controllerクラスの関数のみを記載した）．

細を決定します．これについては「ライン・トレースする」というWhat（何を）の部分と，それを具現化するHow（どのように）の部分をいかに切り離すかが，重要なポイントになってきます．そこで，UMLのユースケース図を使って外部とのインターフェースを整理することにしました（**図4**）．ライン・トレース・ロボットとしてどのような外部とのインターフェースが考えられるのか，どのレベルで切り出すのかなどを整理しました．

● モデリングにはネーミング・センスが重要

外部とのインターフェースが明確になったところで，いよいよモデル本体の分析に移ります．機能分割の方法についてはさまざまなアイデアがあることでしょう．ここでは，筆者らがモデリングにあたり，最も重要視したポイントについて述べます．

機能分割の際に重要になってくるのが，ネーミング・センスです．クラスの名前，状態の名前，変数の名前，そのすべてが重要です．ここでいい加減な名前を付けてしまう

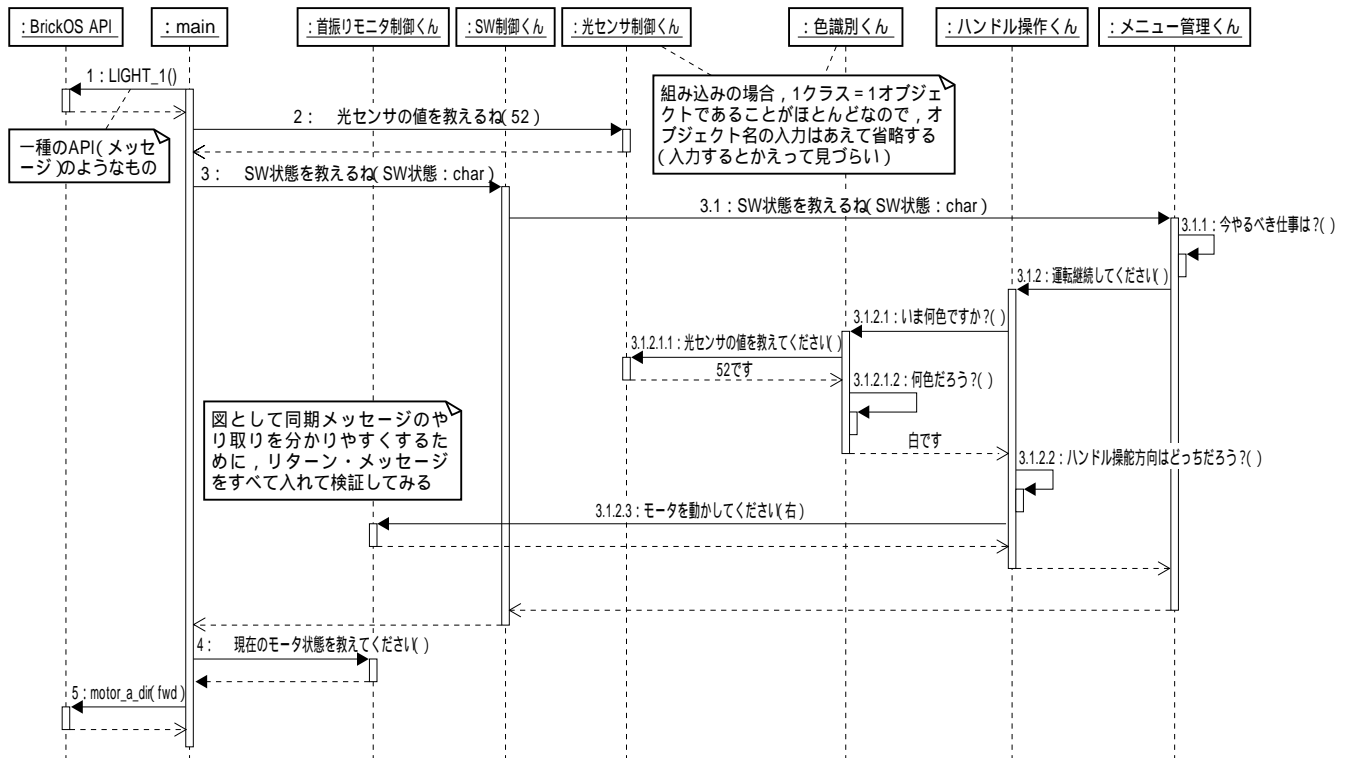



図6　ライン・トレース・ロボットのシーケンス図
「main」や「BrickOS API」などのオブジェクト（インスタンス）がどのようにやりとりするのかを，シナリオ例として時系列で示している．

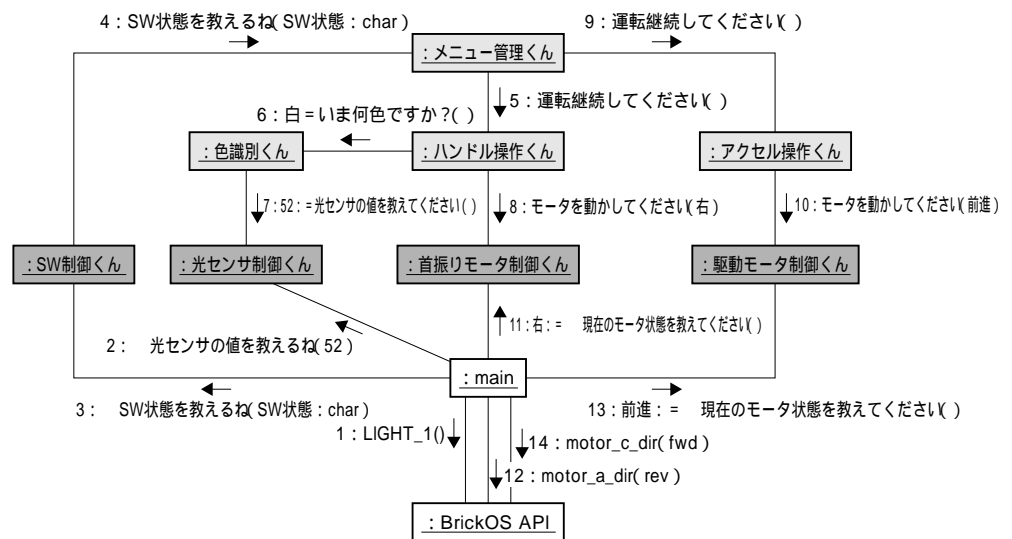



図7
ライン・トレース・ロボットのコラボレーション図
「main」や「BrickOS API」などのオブジェクト（インスタンス）がどのようにやりとりするのかを，メッセージとその順番を含めて示している．

と，たちまち混乱や誤解を生み，モデルがドキュメントとしての機能を失います．良い名前が思いつかない場合は，主に二つの理由が挙げられます．一つは，単純にボキャブラリが少ない場合，そしてもう一つは，その機能がまだまだ複数の機能を複合的に有しており，一つに定められない場合です．機能分割に迷ったら，機能をさらに分割したり，同じような機能を一つにまとめたりといった，機能の統廃合が必要です（**図5**）．

次に，UMLのシーケンス図（**図6**）やコラボレーション図（**図7**）を使って，このモデルが実際に動作するのかどうかを確認しました．具体的には，インターフェースを中心に，次のような点を検証しました．

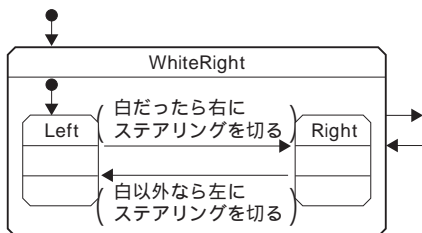



図8　ステアリングのステートチャートの一部（略図）
百聞は一見にしかず．さらに，ステアリングの操舵方向が「左」の状態を左側に，「右」の状態を右側に配置することにより，分かりやすさが増す．

図9
加速/減速のステートチャートの一部（略図）
高速を上側に，低速を下側に配置している．

- 基本動作を行う上で，全体の作業の流れや登場人物（オブジェクト）に過不足はないか．
- メッセージのやり取りは適切か．一連のやりとりの中で頑張りすぎているクラスはないか．
- メッセージの抽象度は適切か．作業負荷に無理はないか．

● MDDツールの変換特性を知る

ロボコン参戦にあたり，筆者らには「UMLツールの良しあしを判断すること（ツールのベンチマーキング）」というミッションが課せられていました．これは，ツールを使うことの効果がどれほどのものかを判断するものです．UMLの図はなんとなく描けたとして，それが実際にどのようなコードに落ちるのでしょうか．

今回使用したツール（Rhapsody）の場合，いくつかの設定を選ぶことができました．今回，「パッケージ」はフォルダとして，「クラス」はファイルとして生成するように設定しました．また，クラスの「属性」は変数として，「操作」は関数としてコード生成されます．さらに，実行環境（brickOS）で使用するmakefileや`main`関数まで自動生成してくれるので，自動コード生成→コンパイル（makefile実行）→実機へのソフトウェアの転送といった一連の作業を1クリックで行え，非常に手軽に作業を進めることができました．

今回のような，自動コード生成が可能なUMLツールを使用する場合，まずクラス図を使って全体の構造を決めることになります．各クラスごとの詳細な動作は，Action言語と呼ばれる言語で記述することになります．Action言語はツールによって異なり，独自の言語仕様だったり，C（C++）言語を使用して記述するものだったりします．独自の言語仕様の場合，C言語で記述すれば数行かかるものを1行で実現できるようですが，それぞれ一長一短があるようです．Rhapsodyの場合，Action言語は（筆者らがC言語用のRhapsodyを使用しているため）C言語となります．新たな言語仕様を覚え直す必要がないので助かりました．

このように，言語記述から完全に解放されることはないのですが，構造や機能（役割）が明確に分割されているため，各クラスでできること（書けること）は自然に制限されてきます．このおかげで，ほかのクラスとの依存を抑止できます．

実は，ETロボコン2006の競技会（タイム・トライアル部門）当日の試走で，筆者らのソフトウェアがまったく動かず，ロボットが数cmも走ることなくコース・アウトするという現実に直面しました．レースを控え，わずかな時間を利用して急きょ修正を強いられたわけですが，このときも機能が明確に分割されていたことによって，どこのクラスのどこの操作を修正すればよいのかがスムーズに突き止められました．また，データ（属性）もカプセル化されていたため，ほかへの影響度も容易に推測でき，修正による副作用を発生させることなく，短時間で対応できました．モデル駆動開発の有用性について，身をもって知ることとなりました．

● モデリングのメリットは視覚的効果にもあり

ソフトウェアの構造について，文章で説明するより図を使ったほうが分かりやすいという例を紹介します．**図8**は，ステアリングのステートチャート（状態図）の一部です．光センサの値が白だったら右に，白以外だったら左にステアリングを切るという指示を出すものです．言葉で書いた説明を読むより図を見た方が，説明する側もされる側もより適切に理解できます．このように視覚的に表現することで，

図10
Simulinkに入力したロボット・モデル
ロボット・モデルは，UMLモデル，タッチ・センサ，光センサ，操舵システム，前進システム，ロボット姿勢と電源などのサブモデルから構成される．

（**a**）走行コース・モデル

（**b**）コースの詳細化：黒線の縁の灰色までモデリング

（**c**）照明変動の詳細化：光反射をドット単位で計算

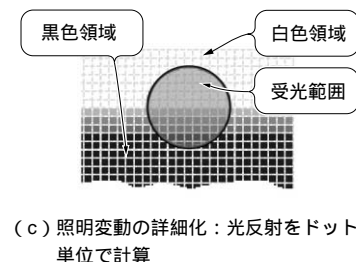
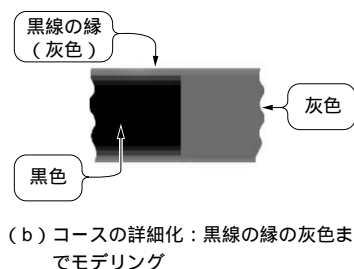
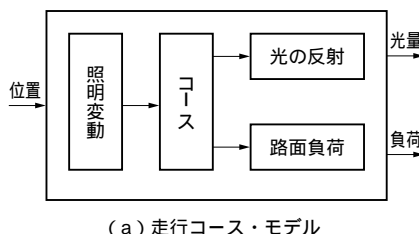
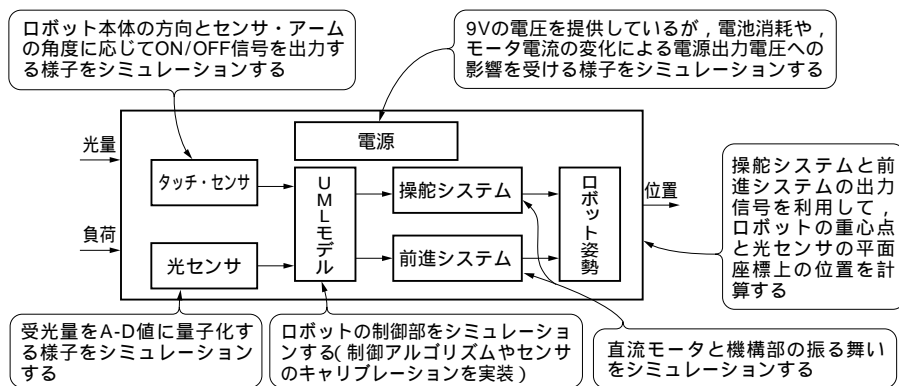



図11　Simulinkに入力した走行コース・モデル
走行コース・モデルは，照明の変動，コース，光の反射と路面負荷などのサブモデルから構成される．

レビューがスムーズかつ的確に進みました．

また，図では空間的なイメージも表現できます．例えば，ステアリングの左右の状態を表現する際に「右」の状態は図中でも右側に配置したり，加速（速度の上昇）/減速（速度の下降）を表現する場合に「高速走行」を上方に配置し，「低速走行」を下側に配置することで，より直感的に理解を促すことができます（**図9**）．

このように，複雑なソフトウェアを単純な機能に分割し，それを視覚的に表現することは，非常に有益だと思われます．

● 細部にわたって現実世界をシミュレーションする

ETロボコンで走らせるLEGOロボットはおもちゃですが，実際の電動自動車と似た機構が多く備えられています．ETロボコンの大会では，あまり精度が良くないと言われている光センサの出力信号を利用して，複雑なコースをトレースしながら自律走行しなければなりません．このような条件でうまく走らせるためには，さまざまな検討が必要です．そのためには，本番と同じようなコース上で走行させ，情報収集や分析，検討を行うという実験的な手法を採ればよいのですが，筆者らが所有している実験設備ではそれは困難です．

そこで，筆者らはシミュレーション・ツールであるSimulinkを利用して，バーチャル・コースとバーチャル・ロボットを作成しました．それを利用して，光センサ信号を分析し，コースをトレースし続ける方法や，前進と操舵システムの制御方法などを検討した上で，UMLモデルに反映しました．また，そのモデルから自動生成したソース・コードをバーチャル環境（SimulinkモデルのS-Function）に実装し，モデルを検証しました．

ETロボコン用のSimulinkモデルは，ロボットと走行コースという二つの物理的なモデルから構成されます．走行コース・モデルは，ロボット・モデルの出力であるロボット重心点と光センサの現在位置信号を入力信号として，光センサの受光量と路面の負荷をロボット・モデルにフィードバックします．

1）ロボット・モデル

ロボット・モデルは，UMLモデル，タッチ・センサ，光センサ，操舵システム，前進システム，ロボット姿勢と電源などのサブモデルから構成されます（**図10**）．また，ロボットの平面運動は，以下の自動車モデルを採用しています．

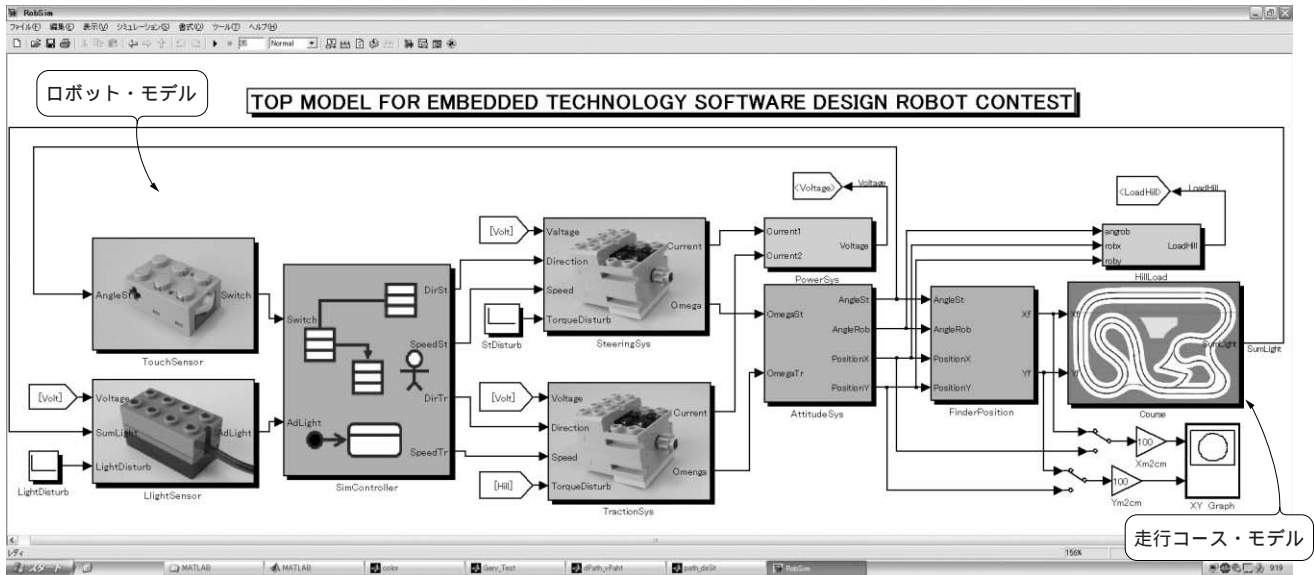



図12　Simulink トップ・モデル図
ロボット・モデルと走行コース・モデルを合わせて実装したSimulink モデル．

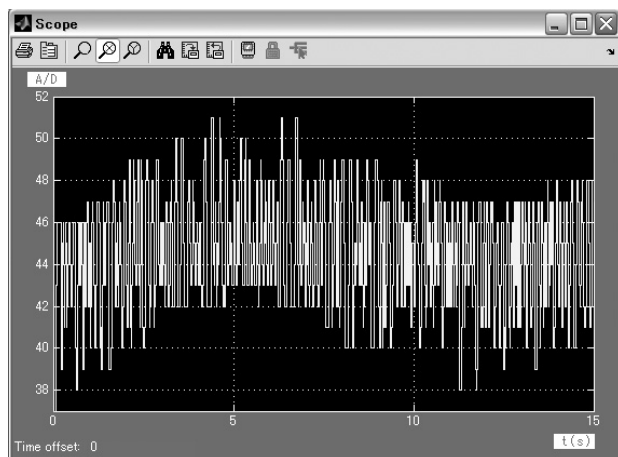



図13　光センサの出力（A-D コンバータ通過後）
作成したSimulink シミュレーション環境を利用して取得した光センサのA-D コンバータ出力値．ロボットの走行位置に応じて照明を変動させている．

$$\begin{cases} mV\left(\dot{\beta}+\gamma\right)=-2K_f\left(\beta+\dfrac{l_f}{V}\gamma-\delta_f\right)-2K_r\left(\beta-\dfrac{l_r}{V}\gamma\right) \\ I\dot{\gamma}=-2l_fKf\left(\beta+\dfrac{l_f}{V}\gamma-\delta_f\right)+2l_rK_r\left(\beta-\dfrac{l_r}{V}\gamma\right) \\ \dot{X}=V\cos\left(\beta+\varphi\right) \\ \dot{Y}=V\sin\left(\beta+\varphi\right) \\ \dot{\varphi}=\gamma \end{cases}$$

この中でX，Y，ϕはロボット重心の平面座標位置と方向を示しています．Vはロボットの前進速度で，δ_fは操舵角度で，そのほかはロボットの固定係数です．

2）走行コース・モデル

走行コース・モデルは，照明の変動，コース，光の反射と路面負荷などのサブモデルから構成されます［**図11（a）**］．コースの形状は，実際のコースの1mm×1mmを1ドットでシミュレーションし，3640×5460のビット・マップ図を作成しました．また，各ドットの明るさ（グレー・スケール）を1バイトで表示しました（その値の大きさは光の反射強度に相当する．例えば，黒：0x00，グレー：0x80，白：0xFFのように保存している）．さらに，ラインの縁において，黒色と白色の間に生じる灰色の領域も考慮し，灰色の縁を設けています［**図11（b）**］．

照明変動モデルは，コース全体の照明が必ずしも均一ではない本番の環境を考慮し，基準のグレー・スケールが座標位置に応じた照明強度によって変動するようにしています．

光の反射モデルは，光センサの受光面積を**図11（c）**のように示す円形と仮定します．光センサの現在位置により決められた円形領域内のドットの光反射強度の合計値を，すべて白色の場合の合計値と比較し，0％～100％の光反射強度に変換します．さらに，照明変動モデルにより得られた現在位置のグレー・スケール値に基づいて光反射強度を光センサ受光量に変換します．

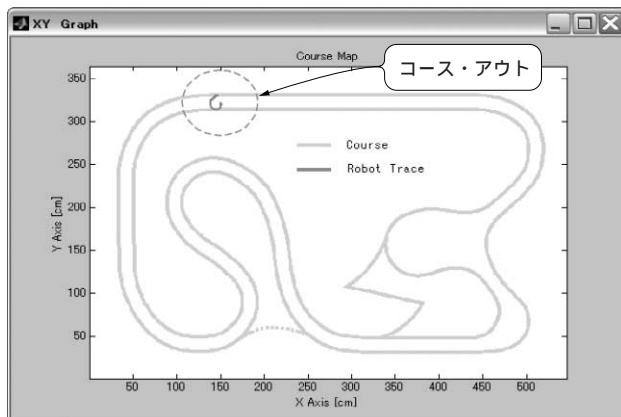
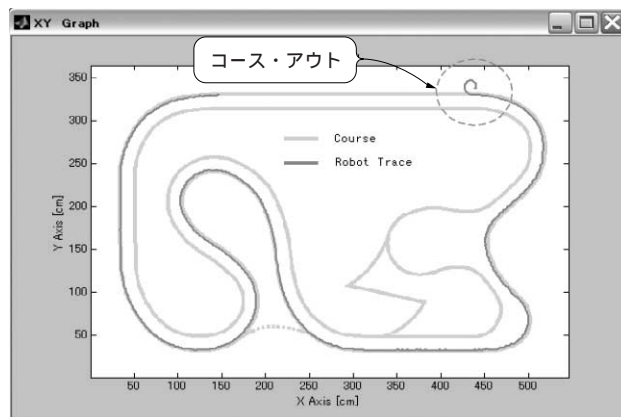
路面負荷モデルは，ETロボコンのコース内にある「坂道」（ゴール・ゲートの手前に配置されている）をシミュレー

(a) 試走時：スタート直後でコース・アウト

(b) モデル修正後：最終コーナーでコース・アウト

図14　実走行の事後シミュレーション
コース・アウト位置がほぼ正しく再現された．

ションするものです．ロボットの現在位置と走行方向などの入力信号を利用して，上りと下りに応じて前進システムへの負荷を変化させます．

これらの各種モデルを合わせてSimulinkモデルの形式に実装したものを，**図12**に示します．また，このシミュレーション環境を利用して，ロボットの走行位置に応じて照明を変動させた場合の，光センサのA-Dコンバータ出力値を**図13**に示します．

さらに，このシミュレーション環境を利用して，ETロボコン2006の試走のようす［現場の照明変動によりロボットが数cmも走らないうちにコース・アウトした．**図14(a)**］と，緊急修正したモデルの走行の状態［**図14(b)**］を再現しました．これらのコース・アウト位置は，実際のコース・アウト位置とほぼ同じ場所であり，Simulinkモデルの精度は実用可能なレベルに達していると思います．この環境を利用して，筆者らはロボット走行中の負荷の変動や照明の変動，電源変動，操舵と前進駆動の関係などを確認し，コースの特別区間（いわゆる“難所”）への対応方法などを検討しました．

筆者らは，ETロボコン2006を通じ，いろいろなノウハウを学ぶことができました．適材適所でモデル駆動開発を使いこなせるようになれば，ソフトウェアの品質だけではなく開発効率の面でも大きな効果が期待できるでしょう．今後ともその目標にむかって努力したいと思います．

参考・引用*文献

(1) ETソフトウェアデザインロボットコンテスト競技規約，Ver.7.7，2006年5月，http://www.etrobo.jp/Regulation.htm

さとう・まさひろ，しみず・まさひさ，みやこ・いくやす，はやさか・さとし
アルプス電気（株）　車載電装事業部 ファームウエア技術部

<筆者プロフィール>
Teamふるかわ．初めてETロボコンに参加し，モデル部門で審査員特別賞を受賞した．チーム名は，筆者らが働いている場所である宮城県の旧古川市（現在は大崎市）に由来する．